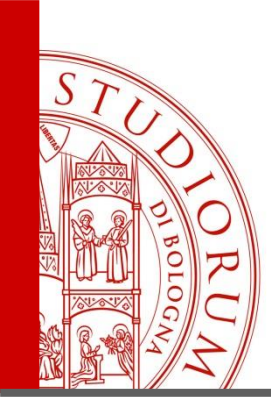


Università degli Studi di Bologna
Corso di Laurea in Ingegneria Informatica

Modelli, Linguaggi e Modelli di Processo di sviluppo

Ingegneria del Software T

Prof. MARCO PATELLA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Sommario

- Modelli
- Linguaggi di Modellazione
- Modelli di Processo

Modelli

Modello

Per ***modello*** si intende genericamente
una rappresentazione di un oggetto o di un fenomeno reale
che riproduce caratteristiche o comportamenti ritenuti
fondamentali per il tipo di ricerca che si sta svolgendo

Modello

- Nei processi di costruzione del software, il termine *modello* va inteso come

Un insieme di concetti e proprietà
volti a catturare aspetti essenziali di un sistema,
collocandosi in un preciso spazio concettuale

Modello

- Per l'ingegnere del software quindi un *modello* costituisce una *visione semplificata* di un sistema che rende il sistema stesso
  - più accessibile alla comprensione e alla valutazione
  - facilita il trasferimento di informazione e la collaborazione tra persone

Modelli & Processi Software

- Nel processo di produzione del software il lavoro dei diversi attori (analisti, progettisti, sviluppatori, etc.) si basa su un insieme di conoscenze che spesso ***rimangono implicite*** all'interno del processo
- Uno degli scopi dei ***processi model-based*** è rendere esplicite queste conoscenze attraverso la costruzione di ***diagrammi*** espressi con notazioni formali…
- … cioè attraverso l'uso di un ***linguaggio*** con sintassi e semantica ben precise

Modelli & Processi Software

- Lo scopo di questi digrammi è rappresentare ***modelli del sistema***
  - per descrivere in *modo conciso e preciso* conoscenze sul problema
  - utili per individuare rischi e scelte progettuali
- L'uso dei modelli è motivato dalla difficoltà della mente umana di impostare ragionamenti efficaci in presenza di dettagli troppo minuti
- I linguaggi per la descrizione dei modelli rappresentano il culmine della continua evoluzione verso ***livelli di astrazione più elevati*** rispetto alle macchine che hanno caratterizzato la storia dei linguaggi di programmazione

Modelli

- L’insieme dei modelli che descrivono un sistema dovrebbe formare una descrizione *completa*, *consistente* e *non troppo ridondante*:
  - la transizione tra un modello e l’altro deve essere continua
  - i modelli connessi tra loro in modo sistematico: un elemento in un modello deve avere il suo o i suoi corrispettivi in un altro

Tracciabilità

- **Traceability**

In qualsiasi direzione si percorra la sequenza
di modelli generati, deve essere possibile mappare
uno o più elementi in un modello in uno o più elementi
in un altro

Tracciabilità

- Nei modelli software, tracciabilità per
  - garantire coerenza e consistenza tra i modelli
  - creare idealmente un percorso logico che parte dai requisiti e arriva al relativo codice (e viceversa)
  - tenere sotto controllo le modifiche
- È spesso molto arduo poter garantire la tracciabilità

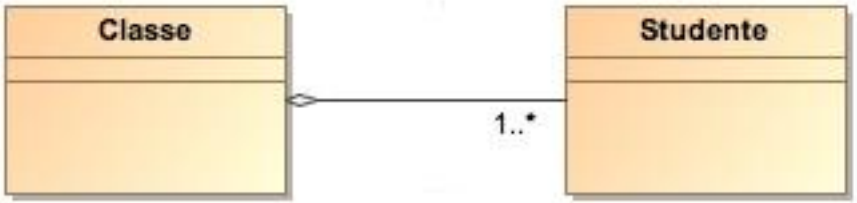
Linguaggi di Modellazione

Linguaggi di modellazione

- Un **linguaggio di modellazione** è un ***linguaggio (semi-) formale*** che può essere utilizzato per descrivere (modellare) un sistema di qualche natura
- Nell'ingegneria del software un modello di un sistema software, o di qualche suo aspetto, prende il nome di ***modello software***
- Quello che noi esprimiamo attraverso i diagrammi è quindi una rappresentazione del modello creata attraverso l'uso di un linguaggio
- Lo stesso modello può essere rappresentato da linguaggi diversi, il cui potere espressivo potrebbe essere differente…

Modelli e Linguaggi: UML

- Come modellereste la seguente frase espressa in linguaggio naturale?
  - “Questa classe è composta da molti studenti”



- Non è l’unica rappresentazione possibile!!

Modelli e linguaggi: XMI

<uml:Model xmi:id='eee_1’>

<packagedElement xmi:type='uml:Class’ name='Classe’ xmi:id=‘527’
visibility='public'><ownedAttribute xmi:type='uml:Property'
visibility='private’ aggregation='shared’ association=’526'/>

</packagedElement>

<packagedElement xmi:type='uml:Class' name='Studente' xmi:id= ‘528’
visibility='public'><ownedAttribute xmi:type='uml:Property'
visibility='private' association=’526'/> </packagedElement>

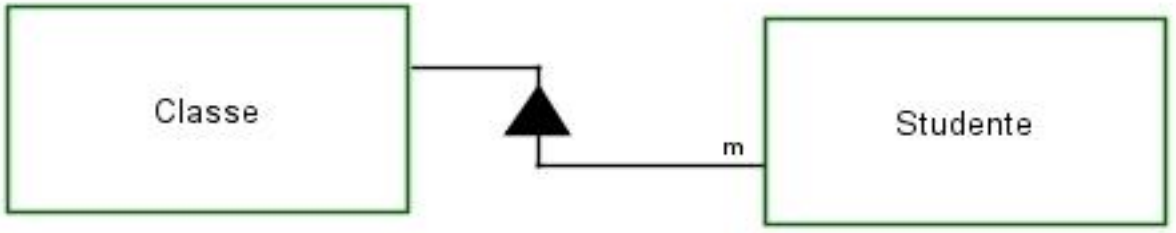
<packagedElement xmi:type='uml:Association' xmi:id=’526'
visibility='public'> <memberEnd xmi:idref=‘527'/> <memberEnd
xmi:idref=‘528'/>

</packagedElement>

</uml:Model>

Modelli e Linguaggi: OPM

OPD



OPL

Classe consists of many Studente.

Modelli e Linguaggi: Codice

- Anche il codice è *una rappresentazione del modello* espressa in un particolare linguaggio
- Il codice:
  - è il modello più dettagliato
  - fornisce una visione “piatta”
  - non mette in evidenza i punti salienti
  - non aiuta ad avere una ***visione d’insieme*** del sistema con un solo colpo d’occhio

Modelli e Codice

- Con le attuali tecnologie a disposizione abbiamo la necessità di creare
  - sia modelli ad alto livello di astrazione
  - sia il codice
- Questo però crea un problema di fondo

I modelli ed il codice sono tipicamente disallineati

Modelli e Codice

- Il disallineamento si crea già durante la fase di implementazione
- Alcune modifiche fatte nel codice non vengono quasi mai riflesse nei modelli di progettazione del sistema
- Tali modelli non offrono più una visuale coerente e viene meno il requisito di tracciabilità
- Operare con tecniche di *Reverse Engineering* causa solo più mali…

Modelli e Codice

- … generare modelli del sistema partendo dal codice produce modelli che presentano tutti i "***difetti***" già riscontrati nel codice
- Fanno perdere l'utilità dei modelli di progettazione
- L'approccio migliore per mantenere l'allineamento e la tracciabilità è
  - apportare le modifiche direttamente al modello di progettazione
  - ove possibile, generare codice da questo attraverso opportuni generatori di codice
  - negli altri casi, modificare direttamente a mano il codice coerentemente con il modello

Linguaggi di Modellazione

- Modelli semantici dei dati
  - Entità-Relazioni (E-R)
- Modelli orientati all'elaborazione dati
  - Diagrammi di Flusso dei Dati (*Data-Flow Diagrams*, DFD)
- Modelli orientati alla classificazione
  - Modelli orientati agli oggetti
- Modelli operazionali
  - Automi a stati finiti
  - Reti di Petri
- Modelli descrittivi
  - Logica del primo ordine
  - Logica temporale

Linguaggi di Modellazione

"Nessun cliente vi ringrazierà per avergli fornito ***diagrammi accurati****;*
quello che vorranno da voi è ***software funzionante***"

- Perché usare un linguaggio di modellazione?
- Dobbiamo risolvere un ***problema di comunicazione***
  - tra i progettisti
  - tra i progettisti e il cliente
  - tra i progettisti presenti e i progettisti futuri…

Linguaggi di Modellazione

- Il linguaggio naturale è troppo impreciso
- Il codice è preciso, ma troppo dettagliato
- La soluzione migliore è utilizzare un linguaggio di modellazione
  - Sufficientemente **preciso**
  - **Flessibile** dal punto di vista descrittivo per poter arrivare a un qualunque livello di dettaglio
  - Possibilmente **standard**

Linguaggi di Modellazione

- Esiste ormai una convergenza su un unico linguaggio di modellazione di tipo grafico: **UML** (*Unified Modeling Language*)
- Sviluppato verso la metà degli anni ‘90 da Grady Booch, James Rambaugh, Ivar Jacobson
- Nel 1999, OMG (*Object Management Group*) ha definito la versione 1.3 del linguaggio (http://www.omg.org)
- La versione corrente è la 2.5.1 (Dicembre 2017) (http://www.omg.org/spec/UML/About-UML/)

Modelli di Processo

Processo di Sviluppo

- Un processo di sviluppo è un insieme ordinato di passi che coinvolge tutte quelle attività, vincoli e risorse necessari per produrre il desiderato output a partire dall'insieme dei requisiti in ingresso
- Tipicamente un processo è composto da differenti fasi messe in relazione una con l'altra
- Ogni fase identifica una porzione di lavoro che deve essere svolto nel contesto del processo, le risorse che devono essere utilizzate e vincoli a cui si deve "obbedire"
- Ogni fase inoltre può essere composta di più attività che devono essere messe in relazione tra loro

Processo Software

- Il processo di sviluppo software è un insieme coerente di:
  - politiche
  - strutture organizzazionali
  - tecnologie
  - procedure
  - deliverable

  che sono necessari per
  - concepire
  - sviluppare
  - installare
  - manutenere

  un prodotto software

Generiche Fasi

- *Specifica*: cosa il sistema dovrebbe fare
  e vincoli di sviluppo
- *Sviluppo*: produzione del sistema software
- *Validazione*: testare che il sistema sviluppato
  sia quello che il committente voleva
- *Evoluzione*: cambiamenti nel prodotto in accordo
  a modifiche dei requisiti o incremento delle funzionalità
  del sistema

Modelli di Processo

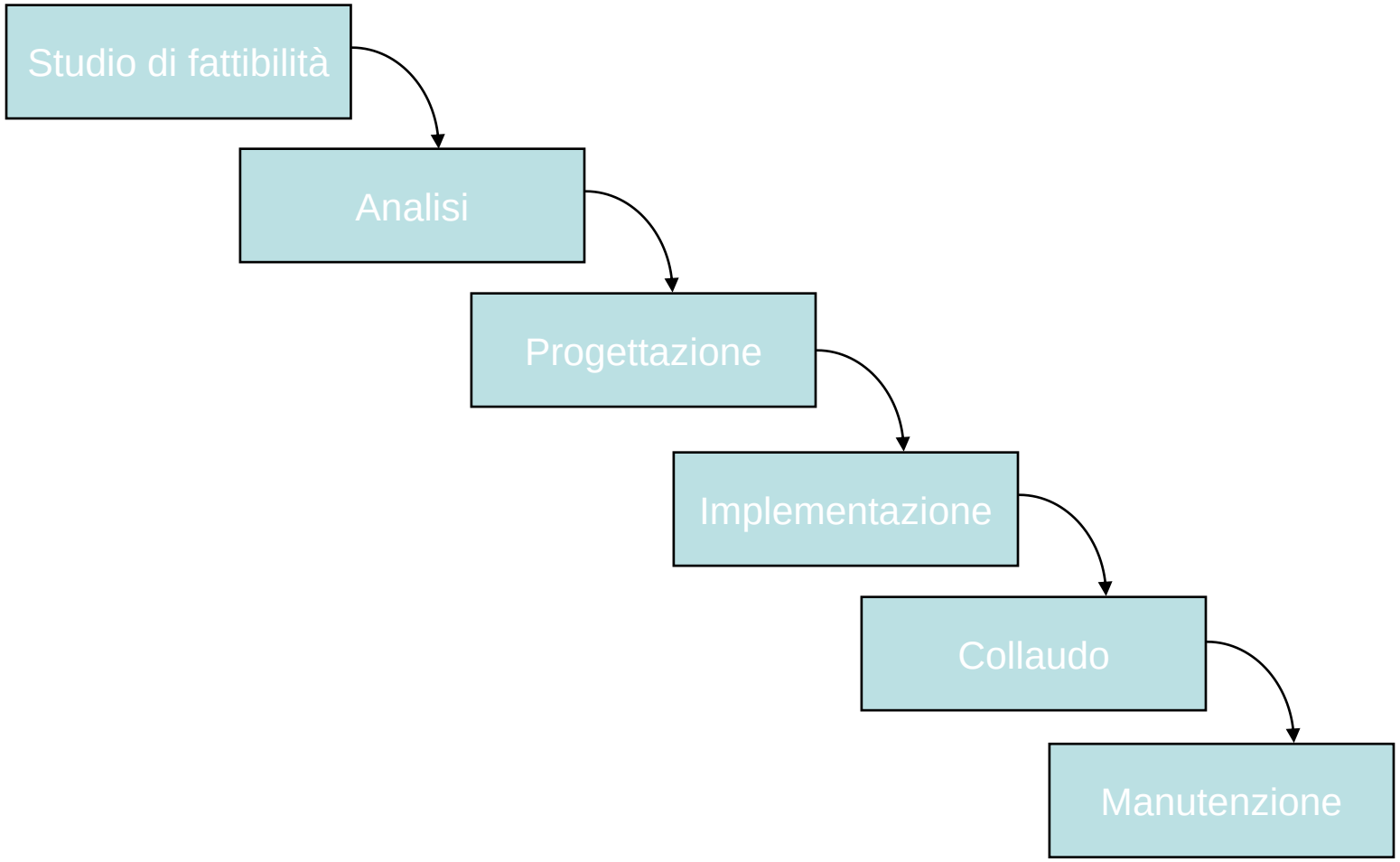
- Un ***modello di processo software*** è
  una rappresentazione semplificata
  di un processo presentato da una specifica prospettiva
- Un modello di processo prescrive:
  - le fasi attorno alle quale il processo
    dovrebbe essere organizzato e l'ordine di tali fasi
  - l'interazione e la coordinazione del lavoro
    tra le diverse fasi
- In altre parole, un modello di processo definisce
  un ***template*** attorno al quale organizzare e dettagliare
  un vero processo di sviluppo

Modelli di Processo di Sviluppo

- Modello a cascata
- Modelli evolutivi
- Sviluppo incrementale–iterativo
- Modello a spirale
- Modelli specializzati
  - Sviluppo a componenti
  - Modello dei metodi formali
  - Sviluppo aspect-oriented
  - Sviluppo model driven
  - *Unified Process* (UP – RUP)

Modello a cascata
(waterfall model)

- Fasi distinte, in cascata tra loro con retroazione finale



Modello a cascata

- Il modello si fonda sul presupposto che introdurre cambiamenti sostanziali nel software in fasi avanzate dello sviluppo ha costi troppo elevati pertanto, ogni fase deve essere svolta in maniera esaustiva prima di passare alla successiva, in modo da non generare retroazioni
- Le uscite che una fase produce come ingresso per la fase successiva sono i cosiddetti semilavorati del processo di sviluppo:
  - Documentazione di tipo cartaceo
  - Codice dei singoli moduli
  - Sistema complessivo

Modello a Cascata

- Oltre alle fasi, è fondamentale definire:
  - Semilavorati
    al fine di garantire che ci possa essere
    un'attività di controllo della qualità dei semilavorati
  - Date
    entro le quali devono essere prodotti i semilavorati
    al fine di certificare l'avanzamento del processo
    secondo il piano stabilito

Modello a Cascata

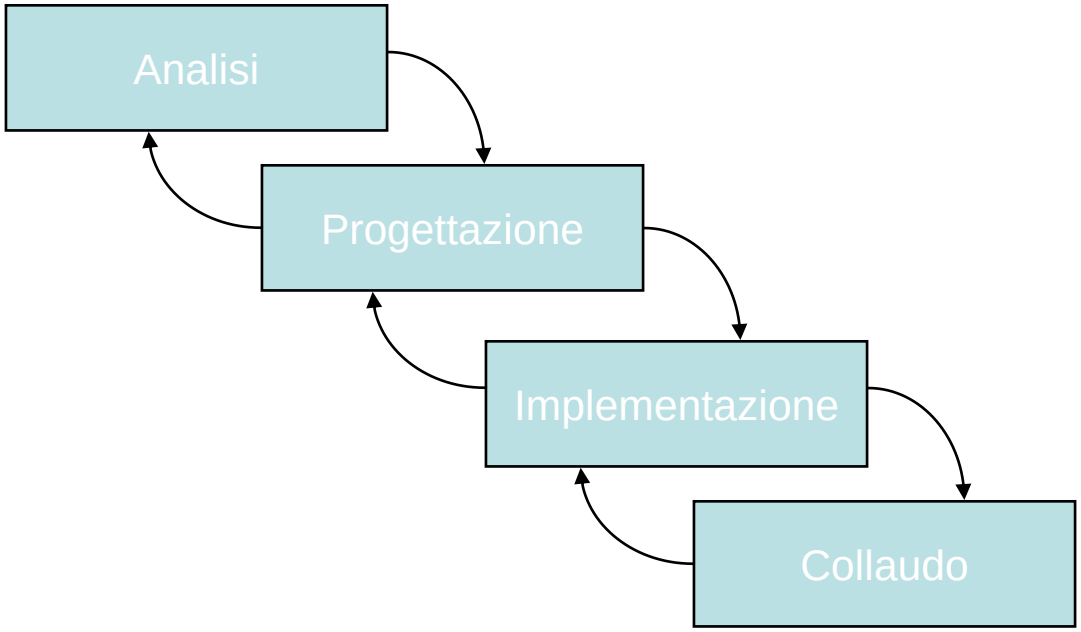
- I limiti sono dati dalla sua rigidità in particolare da due assunti di fondo molto discutibili:
  - Immutabilità dell'analisi
    i clienti sono in grado di esprimere esattamente le loro esigenze e, di conseguenza, in fase di analisi iniziale è possibile definire tutte le funzionalità che il software deve realizzare
  - Immutabilità del progetto
    è possibile progettare l'intero sistema prima di aver scritto una sola riga di codice

Modello a Cascata

- Nella realtà:
  - La visione che i clienti hanno del sistema evolve man mano che il sistema prende forma e quindi le specifiche cambiano in continuazione
  - Nel campo della progettazione "le idee migliori vengono in mente per ultime", quando si comincia a vedere qualcosa di concreto
  - Inoltre, problemi di prestazioni costringono spesso a rivedere le scelte di progetto

Modello a Cascata

- Evoluzioni successive al modello originale ammettono forme limitate di retroazione a un livello



Modello a Cascata

- Per evitare problemi, prima di iniziare a lavorare sul sistema vero e proprio è meglio realizzare un prototipo in modo da fornire agli utenti una base concreta per meglio definire le specifiche
- Una volta esaurito il compito, il prototipo viene abbandonato (throw-away prototyping) e si procede a costruire il sistema reale secondo i canoni del modello a cascata
- Questo approccio risulta però quasi sempre così dispendioso da annullare i vantaggi economici che il modello a cascata dovrebbe garantire

Prototipo

- Modello approssimato dell'applicazione
  Obiettivo: essere mostrato al cliente – o usato da questi – al fine di ottenere un'indicazione su quanto il prototipo colga i reali fabbisogni
- Deve essere sviluppabile
  - in tempi brevi
  - con costi minimi
- Alternativa interessante in tutti i casi in cui lo sviluppo dell'applicazione parta inizialmente con requisiti non perfettamente noti o instabili

Prototipo

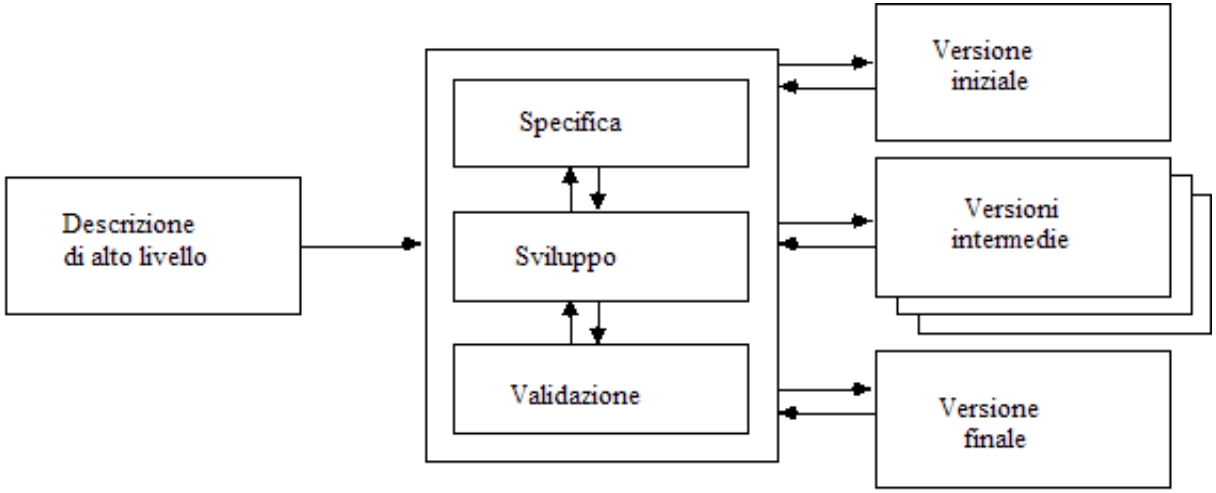
- Prototipazione "*throw-away*"
  - Il prototipo che si realizza è del tipo *usa e getta*
  - Obiettivo: comprendere meglio le richieste del cliente e quindi migliorare la definizione dei requisiti del sistema
  - Il prototipo si concentra sulle parti che sono mal comprese con l'obiettivo di contribuire a chiarire i requisiti

Prototipo

- Programmazione esplorativa
  - Il prototipo si trasforma progressivamente nel prodotto finale
  - Obiettivo: lavorare a stretto contatto con il cliente per
    - Chiarire completamente i requisiti
    - Giungere a un prodotto finale
  - Prima, si sviluppano le parti del sistema che sono ben chiare (requisiti ben compresi)
  - Quindi, si aggiungono nuove parti/funzionalità come proposto dal cliente

Modelli Evolutivi

- Partendo da specifiche molto astratte, si sviluppa un primo prototipo
  - da sottoporre al cliente
  - da raffinare successivamente



Modelli Evolutivi

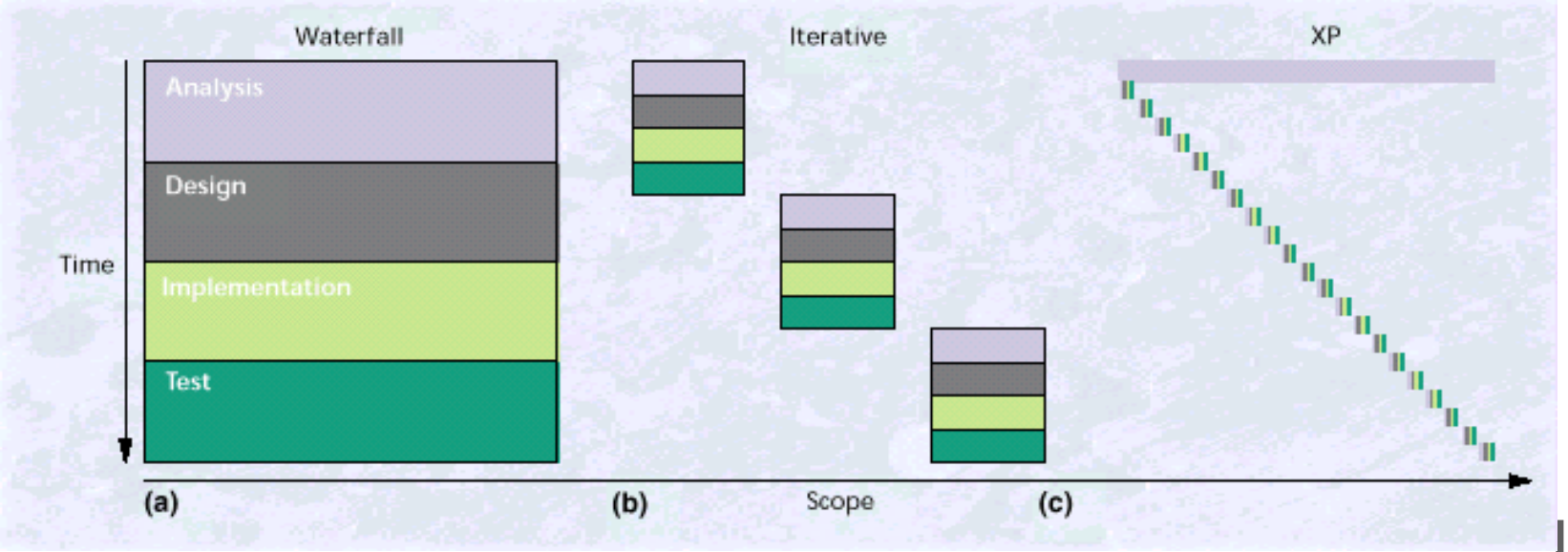
- Esistono diversi modelli di tipo evolutivo, ma tutti in sostanza propongono un ciclo di sviluppo in cui un prototipo iniziale evolve gradualmente verso il prodotto finito attraverso un certo numero di iterazioni
- Il vantaggio fondamentale è che ad ogni iterazione è possibile
  - confrontarsi con gli utenti per quanto riguarda le specifiche e le funzionalità (raffinamento dell'analisi)
  - rivedere le scelte di progetto (raffinamento del *design*)

Modelli Evolutivi

- I modelli evolutivi si sono orientati verso cicli sempre più brevi e iterazioni sempre più veloci, fino ad arrivare al modello più “radicale” che prende il nome di *Extreme Programming* (XP)

Modelli Evolutivi

- L'illustrazione, tratta da un articolo di Kent Beck, mostra l'evoluzione dal modello a cascata, all'*extreme programming*



Modelli Evolutivi

- Problemi
  - Il processo di sviluppo non è visibile (ad es., documentazione non disponibile)
  - Il sistema sviluppato è poco strutturato (modifiche frequenti)
  - È richiesta una particolare abilità nella programmazione (*team* ristretto)
- Applicabilità
  - Sistemi di piccole dimensioni
  - Sistemi che avranno breve durata
  - Parti di sistemi più grandi

Extreme Programming

- Comunicazione tra gli sviluppatori e tra sviluppatori e clienti
- Testing: più codice per il test che per il programma vero e proprio
- Semplicità: il codice deve essere il più semplice possibile
  - complicare il codice tentando di prevedere futuri riutilizzi è controproducente sia come qualità di codice prodotto, sia come tempo necessario - d'altra parte, il codice semplice e comprensibile è il più riutilizzabile
- Coraggio: non si deve avere paura di modificare il sistema che deve essere ristrutturato in continuazione, ogni volta che si intravede un possibile miglioramento (*refactoring*)

Modelli Ibridi

- Sistemi composti di sotto-sistemi
- Per ogni sotto-sistema è possibile adottare un diverso modello di sviluppo
  - Modello evolutivo
    per sotto-sistemi con specifiche ad alto rischio
  - Modello a cascata
    per sotto-sistemi con specifiche ben definite
- Di norma conviene creare e raffinare prototipi funzionanti dell'intero sistema o di sue parti, secondo l'approccio incrementale-iterativo

Sviluppo Incrementale

- Sviluppo incrementale
  - si costruisce il sistema sviluppandone sistematicamente e in sequenza parti ben definite
  - una volta costruita una parte, essa non viene più modificata
  - è di fondamentale importanza essere in grado di specificare perfettamente i requisiti della parte da costruire, prima della sua implementazione

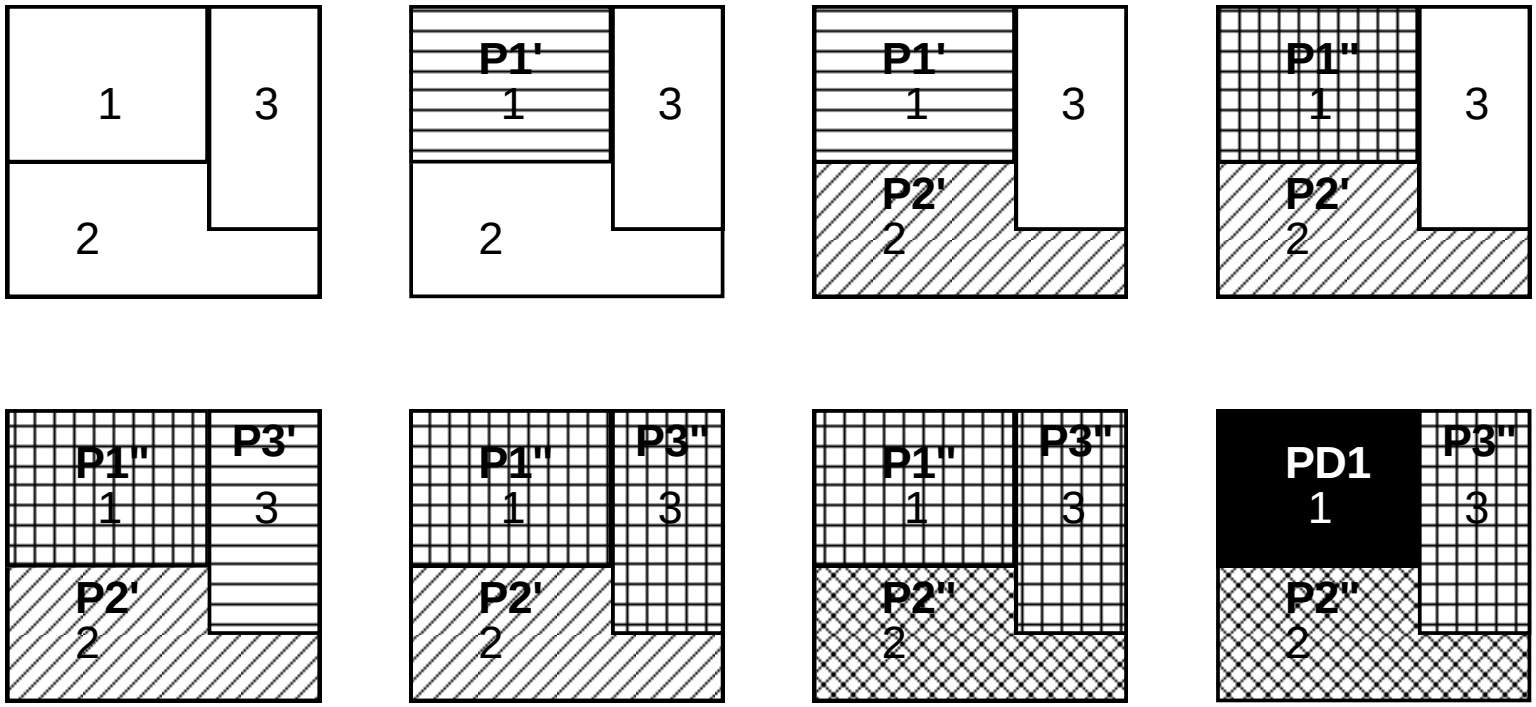
Sviluppo Iterativo

- Sviluppo iterativo
  si effettuano molti passi dell'intero ciclo di sviluppo del software, per costruire iterativamente tutto il sistema, aumentandone ogni volta il livello di dettaglio
  - non funziona bene per progetti significativi

Sviluppo Incrementale-Iterativo

- Sviluppo incrementale-iterativo
  - Si individuano sottoparti relativamente autonome
  - Si realizza il prototipo di una di esse
  - Si continua con altre parti
  - Si aumenta progressivamente l'estensione e il dettaglio dei prototipi, tenendo conto delle altre parti interagenti
  - E così via…

Sviluppo Incrementale-Iterativo

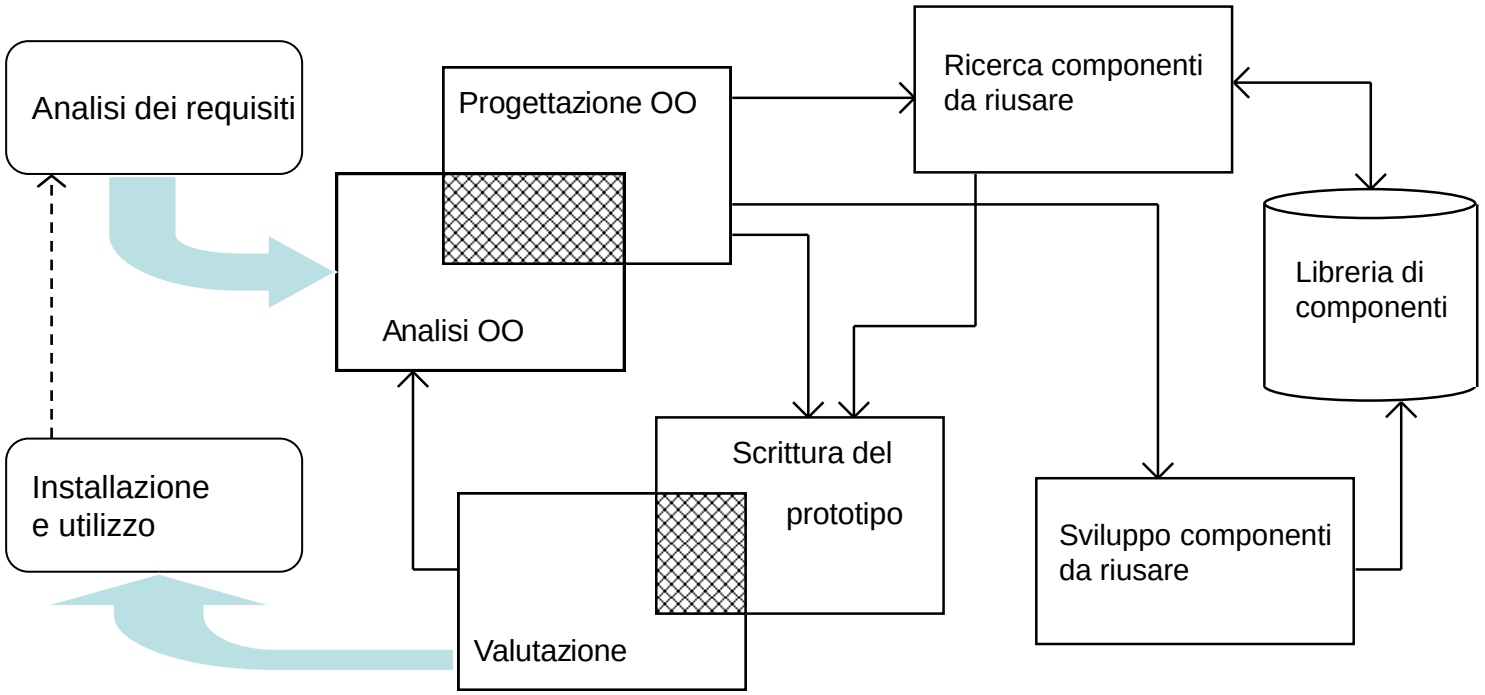


Pn' : Primo prototipo

Pn'' : Secondo prototipo

PDn : Prototipo definitivo

Sviluppo a Componenti



Rational Unified Process

- **Rational Unified Process** (RUP) (estensione dello Unified Process) è un modello di processo software iterativo sviluppato da Rational Software (oggi parte di IBM)
- RUP è un modello ibrido: contiene elementi di tutti i modelli di processo generici
  - non definisce un singolo, specifico processo, bensì *un framework adattabile che può dar luogo a diversi processi in diversi contesti* (per esempio, in diverse organizzazioni o nel contesto di progetti con diverse caratteristiche)
- È pensato soprattutto per progetti di grandi dimensioni

Rational Unified Process

- RUP individua tre diverse visioni del processo di sviluppo:
  - una prospettiva dinamica che mostra le fasi del modello nel tempo
  - una prospettiva statica che mostra le attività del processo coinvolte
  - una prospettiva pratica che suggerisce le buone prassi da seguire durante il processo

Prospettiva Dinamica

- ***Inception (Avvio)*** – generalizzazione dell'analisi di fattibilità
- Lo scopo principale è quello di delineare nel modo più accurato possibile il business case, ovvero
  - comprendere il tipo di mercato al quale il progetto afferisce e identificare gli elementi importanti affinché esso conduca a un successo commerciale
  - identificare tutte le entità esterne (persone e sistemi) che interagiranno con il sistema e definire tali interazioni
  - fra gli strumenti utilizzati ci sono un modello dei casi d'uso, la pianificazione iniziale del progetto, la valutazione dei rischi, una definizione grossolana dei requisiti e così via

Prospettiva Dinamica

- ***Elaboration (Elaborazione)*** – definisce la struttura complessiva del sistema
- Comprende l'analisi di dominio e una prima fase di progettazione dell'architettura
- Devono essere soddisfatti i seguenti criteri:
  - modello dei casi d'uso completo all'80%
  - descrizione dell'architettura del sistema
  - sviluppo di un'***architettura eseguibile*** che dimostri il completamento degli use case significativi
  - revisione del business case e dei rischi
  - pianificazione del progetto complessivo
- Se il progetto non soddisfa i criteri, potrebbe essere abbandonato, oppure rivisitato
- Al termine si transita in una situazione di rischio più elevato, in cui le modifiche del progetto saranno più difficili e dannose

Prospettiva Dinamica

- ***Construction (Costruzione)*** – progettare, programmare e testare il sistema
  - le diverse parti del sistema vengono sviluppate parallelamente e poi integrate
  - al termine della fase si dovrebbe avere un sistema software funzionante e la relativa documentazione pronta
- ***Transition (Transizione)*** – il sistema passa dall'ambiente dello sviluppo a quello del cliente finale
  - vengono condotte le attività di training degli utenti e il beta testing del sistema a scopo di verifica e validazione
  - si deve in particolare verificare che il prodotto sia conforme alle aspettative descritte nella fase di Inception
  - se questo non è vero si procede a ripetere l'intero ciclo

Prospettiva Statica

- La prospettiva statica di RUP si concentra sulle attività di produzione del software chiamate ***workflow***
- RUP identifica sei workflow principali e tre di supporto
- RUP è stato progettato insieme ad UML, quindi la descrizione dei workflow è orientata ai modelli associati ad UML
- Il vantaggio di presentare sia la visione statica che quella dinamica è che le fasi dello sviluppo non sono associati a specifici workflow
  - tutti i workflow *possono essere attivi* in ogni stadio del processo

Prospettiva Statica: Workflow

- ***Modellazione delle attività aziendali***: i processi aziendali sono modellati utilizzando il business case
- ***Requisiti***: vengono identificati gli attori che interagiscono con il sistema e sviluppati i casi d'uso per modellare i requisiti
- ***Analisi e Progetto***: viene creato e documentato un modello di progetto utilizzando modelli architetturali, dei componenti, degli oggetti e sequenziali
- ***Implementazione***: i componenti del sistema sono implementati e strutturati nell'implementazione dei sottosistemi
  La generazione automatica del codice a partire dai modelli velocizza questa fase

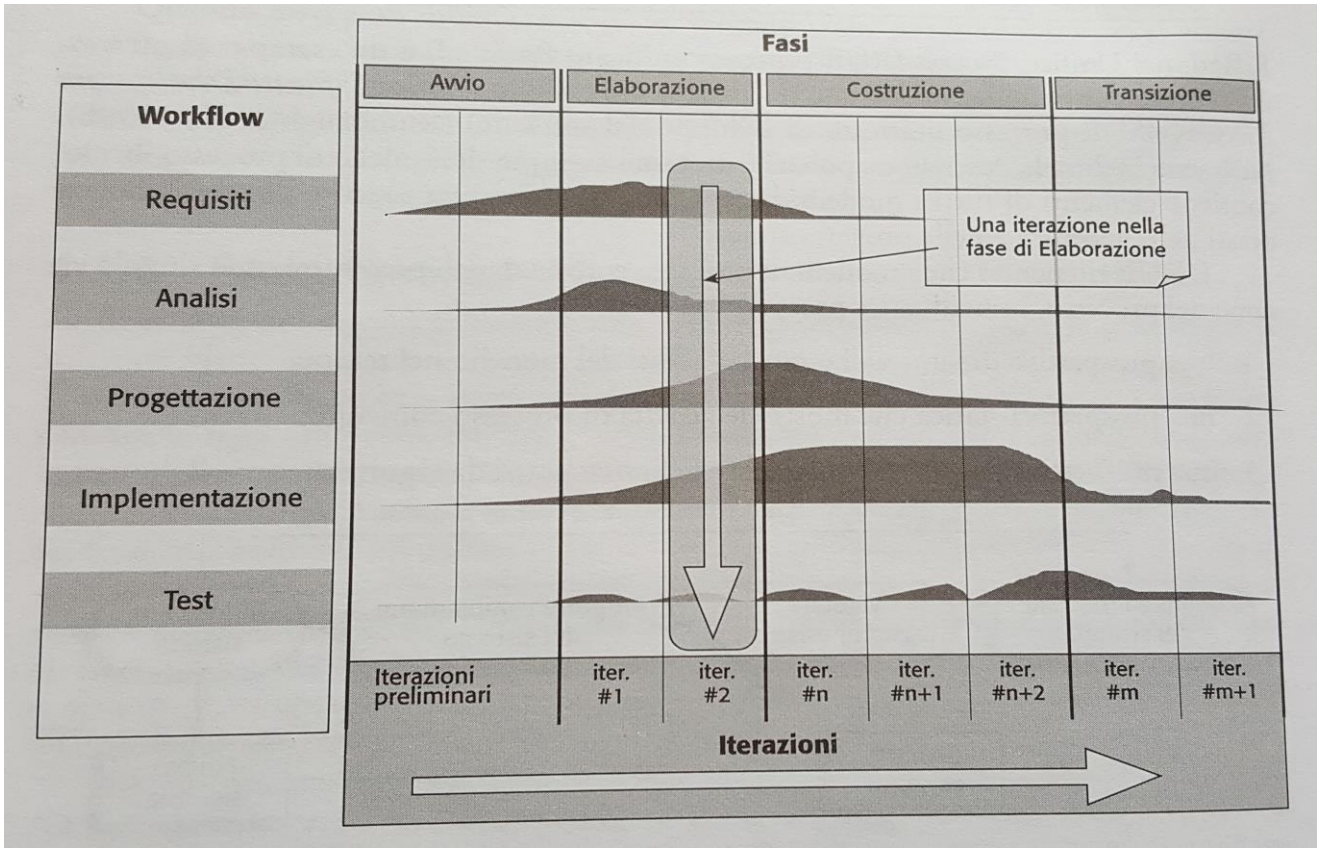
Prospettiva Statica: Workflow

- ***Test***:
  - il test dei sottosistemi è un processo iterativo eseguito in parallelo all'implementazione
  - il test del sistema finale segue il completamento di tutti gli altri test
- ***Rilascio***:
  - viene creata una release del prodotto, viene distribuita agli utenti e installata nelle loro postazioni di lavoro

Prospettiva Statica: Workflow

- ***Gestione della configurazione e delle modifiche***:
  - workflow di supporto che gestisce i cambiamenti nel sistema
- ***Gestione del progetto***:
  - workflow di supporto che gestisce lo sviluppo del sistema
- ***Ambiente***:
  - workflow di supporto che rende disponibili gli strumenti adeguati al team di sviluppatori

Rational Unified Process



- Ogni fase può essere eseguita in modo ciclico con risultati sviluppati in modo incrementale
- L'intero insieme delle fasi può essere eseguito in modo incrementale

Immagine tratta da "Ingegneria del Software" – Ian Sommerville

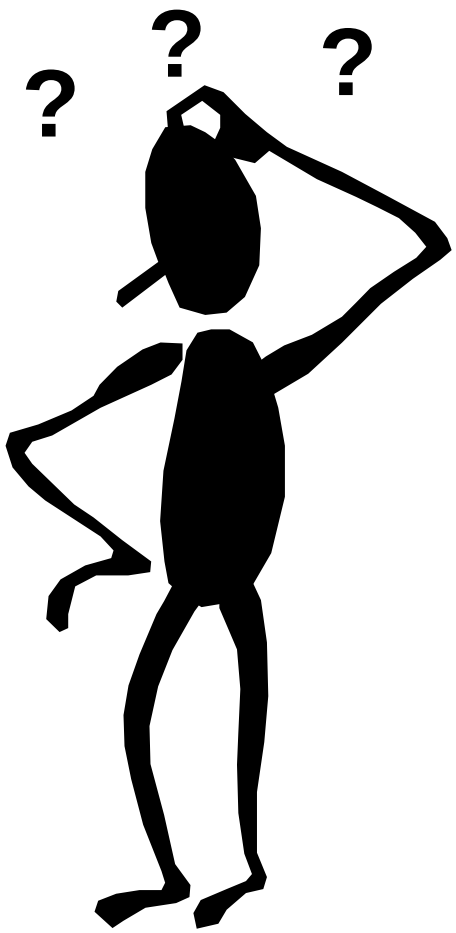
Prospettiva Pratica

- La prospettiva pratica di RUP descrive la buona prassi di ingegneria del software che si raccomanda di usare nello sviluppo dei sistemi
- Le pratiche fondamentali sono sei:
  1. *Sviluppare il software ciclicamente*:
     - Pianificare gli incrementi del sistema basati sulle proprietà del cliente
     - Sviluppare e consegnare le funzioni con la priorità più alta all'inizio del processo di sviluppo
  2. *Gestire i requisiti*:
     - Documentare esplicitamente i requisiti del cliente e i cambiamenti effettuati
     - Analizzare l'impatto dei cambiamenti sul sistema prima di accettarli

Prospettiva Pratica

3. *Usare architetture basate sui componenti*:
   - Strutturare l'architettura del sistema con un approccio a componenti
4. *Creare modelli visivi del software*:
   - usare modelli grafici UML per rappresentare le visioni statiche e dinamiche del software
5. *Verificare la qualità del software*:
   - assicurarsi che il software raggiunga gli standard di qualità dell'organizzazione
6. *Controllare le modifiche al software*:
   - gestire i cambiamenti del software usando un sistema per la gestione delle modifiche, procedure e strumenti di gestione della configurazione

Quale processo



NON ESISTE UN PROCESSO DI SVILUPPO IDEALE

Processi Software

- La definizione di un processo ***non può prescindere*** dalle caratteristiche dell'organizzazione in cui esso avviene e dalla competenza ed esperienza degli attori coinvolti
- Nel 1968 Melvin Conway enunciò una famosa “legge”:

“***Any piece of software reflects the organizational structure that produced it.***”

- proprio per sottolineare che ***esiste uno stretto rapporto*** tra l’organizzazione dello sviluppo e l’architettura dei sistemi che un’organizzazione produce
- Difficile quindi pensare che non vi sia un impatto anche sui processi di sviluppo stessi

Processi Software

- Le fasi e le attività del processo di sviluppo potrebbero essere organizzate in modi differenti e descritte a differenti livelli di dettaglio per differenti tipi di sistemi
- L'uso di un processo inappropriato riduce la qualità e l'utilità del prodotto software che deve essere sviluppato o migliorato

Processi Software

- ***Differenti tipi di sistemi richiedono differenti tipi di processi di sviluppo*** (Sommerville)
- Esempio:
  - Nella progettazione di software realtime per aerei il sistema deve essere completamente specificato e analizzato prima di passare allo sviluppo → modello waterfall
  - In sistemi e-commerce, la specifica e lo sviluppo potrebbero quasi essere parallelizzati → modello evolutivo

…e noi?

- Seguiremo un processo di sviluppo “simil-RUP”:
  - adottiamo una variante di workflow rispetto a quella proposta da RUP:
    - Studio di fattibilità, raccolta e analisi dei requisiti, analisi del problema, progetto, implementazione, test…
  - adottiamo UML come linguaggio di modellazione del software

Il Nostro Processo